EXHITIT A

AdviceNet Overview

Friday, November 22, 1996

Introduction

Computer reliability is like the weather: everybody talks about it, but nobody seems to do anything about it. The reason, perhaps. is that while we sometimes perceive reliability as a s single issue, problems with reliability come from a great many sources, and have an enormous variety of solutions.

In fact, many individual reliability problems have been solved, and more solutions are found each day. Hardware vendors replace faulty parts; software authors revise their work; and users themselves experiment with their machines to find ways to keep them working.

Therefore, we propose to attack the problem of reliability from a different angle — we will treat it not as a software-and-hardware problem, but as a communication problem. Rather than solving problems ourselves, we will provide a way to deliver solutions to the people who need them.

The State of the Art

We are not the first to tackle this communication problem. Most users communicate their solutions informally; many reach a wider audience through user groups and bulletin boards; some sell their advice in newsletters or magazine columns. Technical support workers are paid in part to distribute such advice; system

administrators are paid in part to collect and sift through the advice others have made available.

But these existing pathways of communication have weaknesses we think we can improve upon. Informal communication reaches a limited audience; information is often corrupted as it spreads in this way. User groups, bulletin boards, newsletters, and advice columns require attention even when their advice is not pertinent. Technical support workers are expensive, and have no effective way to broadcast their advice to those who need it. System administrators are even more expensive, and have limited capacity to remember solutions to rare problems.

Our Solution

We propose to create an automated system which uses the internet to distribute advice. Our plan is to build software which establishes a subscription-based, point-to-point network which will deliver signed advice to users in a mixed machine- and human-readable format.

We envision a subscription-based system because we see the relation between an advisor and an end user as an ongoing one. Advisors not only create fresh advice from time to time, but also update and correct their previous advice. We want users to receive these updates with as little effort as possible.

We envision a point-to-point network because there are so many sources of advice that collecting it all would be a monumental task, and because these are so many users that distributing it all would be still more difficult. A point-to-point network distributes this work over many machines on the internet. Further, it gives us a first layer of screening and security: users will (we hope) not subscribe to advice sources which they find irrelevant or untrustworthy. Finally, it allows for privacy: advice can be provided on a part of the internet which is not globally accessible.

We will provide a second layer of security by insisting that advice be signed. The principal defense a user has against bad advice is knowledge of the source: advice from a trusted source is usually good advice. We will build upon that trust by making sure the source is known, and verifying the source cryptographically.

We want at least some portions of the advice to be machine-readable because we want to automatically filter advice for relevancy. We imagine that most advice will concern particular hardware or software configurations, and that most advice will be irrelevant to most users. A significant part of the value we can provide is in locating that portion of the available advice which is relevant. In the ideal situation, the actions advised will also be machine-executable, so that the user will only be called upon to accept or reject the advice.

On the other hand, we realize that neither the conditions under which advice applies nor the advised actions can always be handled exclusively by the computer; even if they could, a user should have the chance to understand the advice before accepting or rejecting it. Therefore it is essential that advice be provided in a human-readable format.

Delivery

We intend to use HTTP as the primary delivery mechanism for advice. Each user will have a list of URIs to be searched for advice; a demon on the user's machine will periodically check those locations for new advice or updates to old advice. The URI will point to a table of available advice, which will contain the URIs and dates for advice files and other advice tables. This nested approach should allow for a relatively small transaction when small changes are made by an advice supplier.

While the user ought to be able to subscribe to an advice site by typing in its URI, we imagine that the subscription will more typically be created by an internet helper program which interprets the URI by adding it to the subscription list. This will make it easy to advertise advice sites on the web or by email. Also, we will want to provide a mechanism by which a program's installer could subscribe the user to the program's official advice site.

We imagine that the delivery demon collects advice and deposits it in a database on the user's machine (perhaps the same demon could be used to make a mirror of advice sites?) In doing so, it should probably make a judgment as to the possible pertinence of the advice — there's no reason for a user's machine to download or remember advice about products the user doesn't own. But advice about things which work now but could fail later should be kept.

Finally, we imagine that users could also get advice from themselves. After putting together a working system, a user could take a snapshot of it — creating an advice database which can locate, if not repair, changes made to the system by by accident. Of course, since users are unlikely to put in the effort to create and keep up to date an entirely reliable advice database, they should take advice from themselves with a grain of salt.

Checking

We imagine a second demon running on the user's machine, which compares the state of the machine to the the advice database. When the conditions attached to a piece of advice are fulfilled, it reports that advice to the user.

The heart of this comparison seems to be a sort of outline-based pattern matching. We imagine that a system can be described as an outline: it has various pieces of hardware and software packages; the packages have folders and files; the devices, folders, and files have various attributes. Advice will come with patterns attached to it; when the system's outline matches the pattern, the advice is relevant.

While the description above covers the intended result, it's probably not the algorithm we want to use. For efficiency we'll want to only generate those parts of the system description which play a part in the pattern matching.

If done in a sufficiently general fashion, we should be able to build other applications on this same core. One which we have imagined is the composition of a personalized newspaper from a set of rules describing a reader's interests.

Reporting

When a piece of advice becomes relevant, we'll prepare a report for the user, containing, or at least summarizing, all of the relevant advice. Our current feeling is that the report will be built from passages of HTML found in the advice, and presented through a web browser. In this way, we get not only a way to include formatted text and pictures, but also links may be embedded which a user can follow for more complete information.

The advice reports may also contain forms, which the user can use to accept machine-executable advice. In some cases, this will require only pressing a button; in others, a more complex form could be presented.

We can create a more flexible reporting mechanism by using a compound-document format such as OpenDoc. This would allow pieces of advice to present richer user interfaces, and may improve communication between our programs and the advice user interfaces.

Taking Action

While the only action necessary in response to some advice is to inform the user, in other cases we imagine that a full response could be available to the user at the click of a button. A web browser acting alone will not make the changes to a user's system which we imagine we'll want, but it is possible that by including a helper program (whose communication with the web browser is secure against off-machine spoofing) we can take more dangerous action, such as downloading and executing a program which will fix a problem.

We expect to include utilities which will, at a minimum, execute the advice users can generate automatically; we may also include utilities for tasks we expect to be advised by third parties, such as upgrading software.